DyVerSe – Dynamic Versatile Semantics

Coordinator: Pierre CLAIRAMBAULT (CR CNRS, LIP, UMR 5668)

DyVerSe aims to develop a theoretical framework for *dynamic/game* semantics for programming languages, capturing in one versatile setting a spectrum of computational features, representative of the heterogeneity of software (*e.g.* higher-order functions, concurrency, probabilities or other quantitative aspects). Our ambition is *(1)* to help unify denotational semantics by providing the missing link between various incompatible models focusing on specific aspects, and *(2)* to provide a toolbox to reason compositionally about the dynamic behaviour of programs, with an eye towards specification and verification.

Partner	Last name	First name	Position	P.M.*	Role
ENS Lyon	Clairambault	Pierre	Chargé de Recherche (CNRS)	38.4	Coordinator
	Laurent	Olivier	Directeur de Recherche (CNRS)	14.4	
	Munch-Maccagnoni	Guillaume	Chargé de Recherche (Inria)	9.6	
	Riba	Colin	Maître de Conférences	7.2	

*P.M. represents the involvement of a member in person.months over the 48 months of the project

1 Context, Positioning, and Objectives

1.1 Objectives and Research Hypothesis

How to *prove* that a program P is correct, or equivalent to an optimized P'? This simple question, prerequisite for certifying software, lies at the heart of decades of work in formal semantics of programming languages. Its study prompted a wealth of developments, each with its methodology and scope. *Operational semantics* axiomatizes execution directly on syntax, while *denotational semantics* gives meaning to programs by embedding them in a suitable syntax-independent mathematical space.

Operational semantics is a powerful and extensible methodology, perfectly fit for formalization in a proof assistant – it is, for instance, behind the celebrated CompCert project [Ler09]. On the other hand, its deployment often follows from ad-hoc choices, and it is not robust to variations in the language. It is tied to syntax and struggles with compositionality[1]. *Denotational semantics* is syntax-independent, and often more principled. It is a great tool to reason about program equivalence (two programs being equivalent if they denote the same object), to prove general properties of languages (*e.g.* termination), and it comes with compositional reasoning principles on programs. The wider mathematical space in which programs are embedded sometimes suggests new useful constructs (it is the birth story of Linear Logic [Gir87]). In exchange, it is more mathematically demanding and it is often quite brittle: distinct fragments of the same language may require radically different representations.

As illustration, we show below a naive probabilistic program for leader election in a distributed system with ring architecture. Each process picks a random identifier, and sends it across the ring. Processes only propagate messages with higher identifier than their own. For succinctness, our program is not robust to collisions of random identifiers. But if they all happen to be chosen distinct, only the largest makes the tour, selecting the leader. The program features higher-order functions, parallelism, side-channel communication[2], and probabilistic choice. It is in an Ocaml-like syntax but all those features exist *e.g.* in Erlang [Arm10]. The command `send(c,m)` sends `m` on channel `c`, while `listen(c)` listens on `c`, returning `m` when received. The function `spawn` takes `N` : `nat` and `f` : `chan` $*$ `chan` $\to$ `unit`, and *(1)* spawns N processes in a ring, each running `f(prev,next)`, with `prev` and `next` new channels linking them to the previous and next process; *(2)* returns the list of all `prev` channels as a means to broadcast the start signal. The parameter `lead` : `chan` $*$ `chan` $\to$ `unit`, is to be ran by the elected leader only.

[1]Operational semantics *can* be made compositional, but behind lie structures from denotational semantics: for instance, the operational semantics behind the recent *Compositional CompCert* [SBCA15] "bears much in common" (quoting the paper) with Ghica and Tzevelekos' operational reconstruction of game semantics [GT12].

[2]Here parallelism means simultaneous execution, independently of communication between threads. Side-channel communications refers to either message passing or shared memory, handled similarly in our semantic framework.

```
let rec proc(id, prev, next, k) =
  if id = 0 then proc(rand_int(),prev,next,k) else                  (* Initialisation   *)
    match listen(prev) with
      | Elect       -> send(next,Candidate id); proc(id,prev,next,k)    (* Sends application *)
      | Candidate n -> if n > id then (send(next,Candidate n);          (* Lost the election *)
                                       proc(id,prev,next,k));
                       if n < id then proc(id,prev,next,k);             (* Drops candidate n *)
                       if n = id then k(prev,next)                      (* Won the election  *)
let main () =
  let threads = spawn(N, fun (prev,next) -> proc(0,prev,next,lead)) in
    map (fun x -> send(x,Elect)) threads                                (* Start election   *)
```

How can we *prove* that for this *program* (as opposed to the abstract algorithm behind), with a certain probability, exactly one process will run `lead`? The combination of features used makes even describing formally the behaviour of `proc` a challenge. On the operational side, this could be done in principle using *Segala automata* [Seg95], which can capture the combination of probabilistic choice with the non-determinism arising from scheduling (though we are not aware of this being done in the presence of higher-order). On the denotational side, there are great models (all *fully abstract*, the ideal match between a language and its semantics) of higher-order languages with parallelism and side-channel communication [GM08], (unobservable) parallelism and probabilities [EPT18], and side-channel communication and probabilities [DH00]; but not to the three together. Even worse, these models have little to do with each other: [DH00] and [GM08] build on *game semantics* but rely on incompatible settings, while [EPT18] belongs to the distinct family of *weighted relational models*. There are no known connections or bridges between them. It is unclear what perspective or information they offer on the full language.

Beyond these specific features, this illustrates the general state of affairs in semantics that DyVerSe aims to address: there is a *missing link* between the syntax-tied operational semantics and a spectrum of incomparable denotational semantics of restricted scope. DyVerSe aims to construct that missing link: a syntax-independent and fine-grained description of program executions which could serve both as a unified compositional operational semantics for a wide array of computing features, and as a bridge between specific denotational models. To achieve this, DyVerSe will build on and extend *concurrent/asynchronous games* [AM99b, RW11], whose recent developments [CGW12, CCW14, CCW15, CCPW18] suggest a plan of attack. Beyond its foundational value and its use as a tool to transport semantic results, we believe this missing link will provide tools to put under scrutiny the structures used to eliminate *race conditions* and ensure harmonious use of resources in languages like *Rust*, lend itself to algorithmic treatment, and offer new opportunities for the certification and verification of programs.

1.2 Position of the Project as it Relates to the State of the Art

Traditional denotational semantics (*e.g.* Scott domains) model programs as functions, through their input/output behaviour. Effects (*e.g.* state, non-determinism, *etc*) can be captured via monads, but monads do not readily combine. Though combining effects has been a driving question in denotational semantics these past decades, it is hardly a streamlined process. For instance, though there is significant recent research activity around domain settings supporting probabilities and higher-order [SYW$^+$16, VKS19], it is unclear how they combine with non-determinism [Gou17], let alone concurrency; nor how all these models relate together.

Game semantics [HO00, AJM00], though also a denotational semantics, takes a radically different approach: instead of a function it represents a program as a *strategy*, *i.e.* the collection of (representations of) its interactions against all execution environments. Once executions are first-class mathematical objects (called *plays*) one can characterise those achievable using only specific effects. This led to a wealth of fully abstract models, rewarded in 2017 by the Alonzo Church Award (from the ACM SIGLOG, the EATCS, the EACSL, and the Kurt Gödel Society). To cite the announcement:

> "Game semantics has changed the landscape of programming language semantics by giving a unified view of the denotational universes of many different languages. This is a remarkable achievement that was not previously thought to be within reach."

But are games models truly "unified"? For *deterministic sequential* programs, yes: various effects (degrees of control and state) are indeed captured as additional conditions on one single canvas [AM99a].

But the very representation picked for executions (plays) impacts expressivity, and imposes constraints on the execution environment. Dealing with effects such as concurrency [GM08], demonic non-determinism [HM99] or exceptions [Lai01] demanded reworking the notion of play (in incompatible ways); while extensions with probabilities [DH00] or non-determinism [HM99] appeared too rough to support a characterization of "pure" executions (*i.e. innocence* [HO00]). Beyond sequential deterministic languages, current game semantics is far from unified, it is – just like semantics in general – a scattering of independent developments made incomparable by their implicit or explicit assumptions on the underlying computational model. The challenge behind DyVerSe is – besides modeling combinations of effects as of yet not covered – to push this unified picture beyond sequential deterministic languages.

To attack this, DyVerSe relies on another family of game semantics, called *concurrent*, *causal* or *asynchronous*. Pioneered by Melliès and others [AM99b, Mel04, MM07, FP09], they have been actively developped recently (with major contributions by the coordinator [CGW12, CCW14, CCW15, CCPW18]), building on new definitions due to Winskel [RW11]. Following a long development, they have only recently reached the maturity to compare in expressivity with traditional models. Concurrent games build on *event structures* [Win86], a notion originating from connections between domain theory and concurrency theory [NPW79]. Event structures are the natural *truly concurrent*[3] analogue of trees, representing causal dependence and independence: for instance Petri nets canonically unfold to event structures [HW08].

It is natural to expect game semantics based on event structures to provide models of concurrent languages. In addition, our recent work [CGW12, CCW14, CCW15, CCPW18] has emphasized a perhaps less expected phenomenon: switching to a causal setting means that strategies are not anymore representations of *how programs are observed by a specified type of environment*, but of *what are the causal dependencies and constraints* behind the program's actions, independently of the execution environment. This has far-reaching implications even for sequential programs: this commitment to specific types of environments is exactly what causes the variety of traditional games models to be incompatible. And indeed, to substantiate this intuition we have recently obtained the first results hinting at our desired unified view: in [CCPW18] we gave the first games model that *(1)* is fully abstract for *pure* probabilistic programs, and *(2)* embeds compositionally into traditional probabilistic game semantics [DH00] and into the weighted relational model [EPT18], two radically different models.

Besides denotational semantics, traditional game semantics have relatively well-studied connections with operational semantics [DHR96, LS14]. However, these work rely on interleaving-based representations of execution, and so fail to apply to our (truly concurrent) event-structure-based setting. So instead, DyVerSe proposes an original approach based on unfoldings of *multi-token geometry of interaction* (first introduced by Laurent [Lau01]), regarded as generalizations of Petri nets.

1.3 Methodology and Risk Management

The core ambition of DyVerSe is to push the development of concurrent games to deliver the truly "unified view of semantics" that traditional game semantics promised, along with a convincing relationship with operational semantics. Beyond this core goal we aim to give the community tools to put our structures at work – for instance for specification and verification purposes; to study semantic structures behind concurrent programming languages (such as *Rust*) ensuring a harmonious interplay of shared resources; or for other situations where a sharp causal understanding of complex computing systems may help, such as in logic and proof theory.

The concrete objectives of DyVerSe span five *tasks*, and each task is composed of three *objectives* or more. For each objective we specify the anticipated duration to reach its main goals (*short-term*, *medium-term* or *long-term*) once its potential dependencies are met; and an assessment of the risk (*low-risk*, *medium-risk* or *high-risk*) – how confident we are that research unfolds as planned. This being fundamental research these elements of foreplanning are, of course, subjective and indicative.

Task A. Structures **(P. Clairambault, PhD student)**

When designing a game semantics, the first choice is: what mathematical structure to use for representations of *plays* and *strategies*? It is a critical one. In traditional game semantics, as strategies describe

[3]They represent concurrency through partial orders rather than interleavings. Truly concurrent models have been advocated for the verification of concurrent systems, as they avoid the *state explosion problem* due to interleavings.

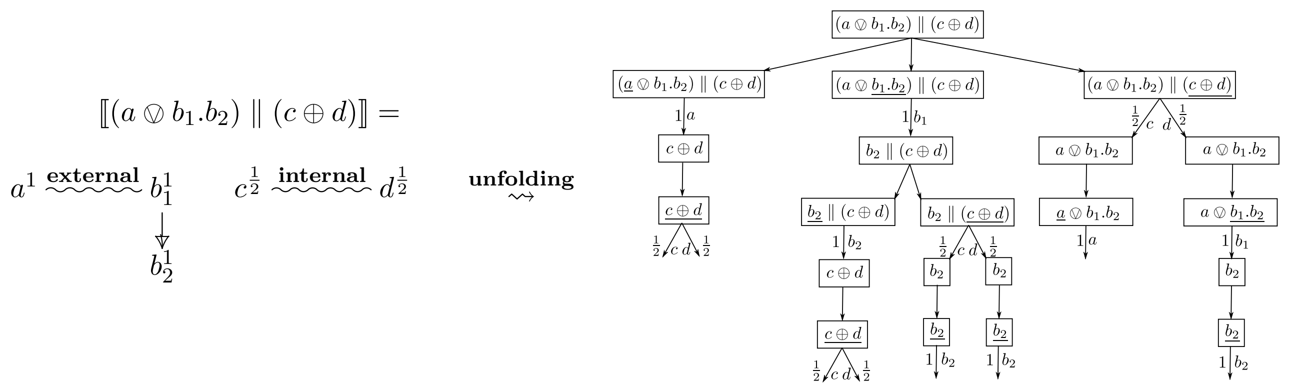

Figure 1: A Segala Event Structure and its Unfolding as a Segala Automaton

how programs are viewed by their execution environments, the choice of the mathematical representation of plays bears a commitment to one ambient programming language outside of which the games model becomes meaningless. Concurrent games avoid this by presenting *causal choices* behind the program's behaviour. However, there are still a wide variety of causal structures that one may adopt, impacting how rich the representation will be. Though the concurrency literature is rich in such structures, for some of our objectives we will need to develop new, more expressive ones.

Task A regroups objectives towards new structures, prerequisite to their application in DyVerSe. Objectives **A.1** and **A.3** aim for more expressive mathematical settings for true concurrency. Those are not specific to game semantics and should impact the wider *concurrency* community. Objective **A.2** studies structures more specific to games.

A.1 Non-determinism and Probabilities. We demonstrated in [CCPW18] that concurrent games are perfectly fit as a model of probabilistic programs (we focus here on discrete probability distributions, though an extension to the continuous case has been proposed [PW18]). However, this concerns only *purely* probabilistic programs, such as purely functional programs extended with probabilistic choice, in which all choices are probabilistic. In the presence of concurrency and side-channel communication (as in our example on p.2), data races are solved externally by the memory or the scheduler, and there appears to be no meaningful way to express this purely probabilistically.

We will extend *probabilistic event structures* [Win13] so that they support non-determinism and probabilities. For that we will first define *alternating* event structures, supporting two kinds of conflict: one *internal* and one *external.* Adding probabilities will be done so that for each resolution of the external conflict, we get a valid probabilistic event structure. We will validate our model by giving an unfolding to tree-like *Segala automata* [Sto02] (a standard to represent systems that combine probabilistic and non-deterministic choice) preserving the interpretation of probabilistic CCS [DD07], thus yielding the *truly concurrent* analogue of Segala automata, in the sense that event structures are the truly concurrent analogue of trees. In Figure 1, we display the envisaged solution. On the right hand side we show the interpretation of a process of probabilistic CCS ($\varovee$ is non-deterministic choice and $\oplus$ is a fair probabilistic choice) as a Segala automaton. The tree alternates between the two kinds of choices: non-deterministic choices resolving a choice of the scheduler, and probabilistic choices sending a signal with a certain probability. On the left hand side we show the corresponding "Segala event structure", with immediate causal dependency written $\rightarrowtriangle$, two kinds of conflicts, and probability annotations on events as a superscript.

We consider this objective as *short-term* and *low-risk*, as investigations are already underway.

A.2 Enriched Concurrent Games. The representation of programs in concurrent games is *extensive*, in the sense that ground data domains are represented in a flat way, aligning all (potentially infinitely many) possible values. Consequently, a program taking an integer as an argument has a different branch for each possible value. In some cases, for instance when the data flow has little impact on the control flow, it is tempting to keep the two separate; representing the data flow as an additional information on top of the causal structure. In [ACHW18], we followed this idea to extract constructive information from classical proofs, and in [ACL19] we extended this to keep track of *time* in the execution of concurrent programs. In both cases the additional information is regarded as *metadata*, that is tracked throughout execution but does not affect the control flow.

The objective of **A.2** is, drawing inspiration from the two works above, a general theory of *enriched concurrent games*, where additional information added to the causal structure comes, for instance, from an arbitrary cartesian category or Lawvere theory. In contrast to [ACHW18] and [ACL19], we want to make it so that the control flow may indeed be affected by this additional data, by cutting events carrying information that is dubbed *inconsistent*. In our opinion, a requirement for such a setting to be successful is the ability to perform *symbolic computation*, where *e.g.* the additional information is composed of sets of formulas such as numerical constraints, cutting out inconsistent branches. We leave however for **E.2** the implementation and application to verification of symbolic concurrent games.

We consider this as *short-term* and *low-risk*; the structures involved are well-understood.

A.3 Nominal Event Structures. *Names* form a special data that can be generated, passed around, and tested for equality, but with no further structure. This is captured by mathematical frameworks such as *nominal sets* [Pit13] that force all operations to behave uniformly *w.r.t.* names, in the sense that their behaviour cannot depend on specific representatives. Nominal techniques have established themselves as a valuable addition to the toolbox of researchers in semantics and verification. Besides their intended use to represent *binding*, they provide means to speak about *locations* (*e.g.* memory locations) or *messages* abstracting away the low-level aspects, leading to applications in *semantics* [MT11] and *security* [Gor00]; they also provide symmetries extending automata techniques to infinite alphabets [Seg06]. In traditional game semantics, their impact has been considerable [MT16].

The main objective of **A.3** is to define and study *nominal event structures*. In nominal event structures, events carry *names*. Events can generate fresh names, but they may also carry names introduced independently, provided their generation occurred in their *causal history* – so that name leakage is constrained by causality. This will be, to our knowledge, the first truly concurrent nominal model, which we will relate to its interleaving counterparts in terms of automata [Tze11] or transition systems. As a side objective, we will investigate the possibility to simplify, through the use of names, our mechanism to handle replication of resources via *symmetry* in concurrent games [CCW14]. If, as we suspect, nominal sets lack expressivity for this purpose, we will see how they can be extended.

We consider **A.3** a *medium-term* and *medium-risk* objective. It is rather open-ended but has a high potential for impact inside and outside DyVerSe.

Task B. Representation (P. Clairambault, O. Laurent, G. Munch-Maccagnoni, Postdoc)

Task B regroups objectives towards the core semantic objective of DyVerSe, *i.e.* the unified framework for denotational semantics. Of course, such an objective needs to be qualified in order to be meaningful. No denotational model can claim to cover the whole realm of programming features, which is wide and open-ended; nor would it be wise to try. By nature, denotational models abstract away some aspects of computation. To take some extreme examples, covering features such as *meta-programming* or *explicit memory management* would question the hypotheses that underlie the very methodology of denotational semantics, and which makes it useful to reason about programs.

More concretely, the approach followed by DyVerSe is to aim for the right balance and attack a combination of features that *(1)* is sufficiently widespread to be significant, and *(2)* addresses some of the fundamental difficulties intrinsic to considering combinations of effects. Following these guidelines, we have identified three effects usually modelled via radically different theoretical tools:

$$\{\textit{parallelism}, \textit{shared state}, \textit{probabilistic choice}\}$$

Drawing inspiration from Abramsky's *semantic cube* research programme [AM99a], we analyse these effects through extensions of PCF [Plo77], Plotkin's paradigmatic pure Call-By-Name programming language. The combinations yield 8 languages (organized in the "DyVerSe cube" in Figure 2), among which 5 already have known fully abstract models: PCF [HO00, AJM00], *Idealized Algol* (IA) [AM96], *Probabilistic* PCF (PPCF) [EPT18], *Probabilistic Idealized Algol* (PIA) [DH00], and *Idealized Parallel Algol* (IPA) [GM08]. The two languages $\mathsf{PCF}_{\parallel}$ and $\mathsf{PPCF}_{\parallel}$ are extensions of PCF and PPCF respectively with a parallel primitive. Finally, *Probabilistic Idealized Parallel Algol* (PIPA) has, to our knowledge, no denotational model. Though this makes it seem like the situation is well-explored, as argued before these fully abstract models are radically different, and in most cases have no clear relationship with each other.

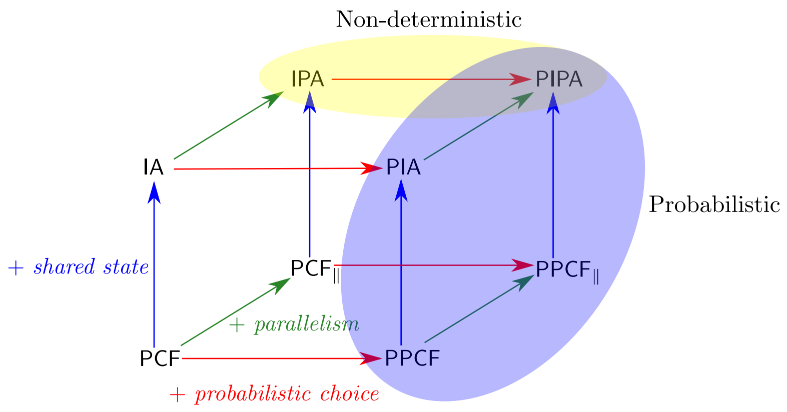

Figure 2: The DyVerSe cube

However, their existence provides us with clear targets and points for comparison, in the sense that our unifying framework should have a clear connection with them all.

The reader might wonder why we single out these particular features, whereas there are multiple others around: control [Lai97], exceptions [Lai01], or even quantum primitives [CdVW19] to cite a few. Recall that game semantics was celebrated for its ability to accommodate many computational effects in a single framework, *for deterministic sequential programs*. The fact that this failed to extend to the non-deterministic case was noticed early on [Har99], and attributed to the lack of *explicit branching structure* – this analysis was confirmed quite recently, independently by Tsukada *et al* [TO15] and ourselves [CCW14]. From this experience stems our analysis: *effects* combine in subtle ways with *branching behaviour*, and so a successful unifying framework must first represent branching adequately.

Amongst our three effects, *parallelism* and *probabilistic choice* both embody a form of branching of the execution. The third effect, *shared state*, is not about branching *per se*, but by providing *side-channel communication* it interacts with parallelism and generates a third kind of branching, *non-determinism*, as resolution of data races corresponds to an external choice which cannot be sensibly resolved by assigning probabilities[4]. We believe that dealing adequately with those three types of branching is the main obstacle to a successful unifying framework, and that once this is done, other non-branching effects (message-passing, control, exceptions, *etc*) can be handled more directly by integrating the relevant structures from sequential games.

B.1 Full Abstraction for PIPA. Amongst the 8 languages of the DyVerSe cube, 5 already have fully abstract models. But in fact PCF$_\parallel$ and PPCF$_\parallel$ do as well: without side-channel communication parallel execution is non-observable, and can be sequentialized without affecting observational equivalence. So, up to observation, these two languages do not significantly differ from their sequential counterparts.

Therefore, preparing the ground for the DyVerSe cube leaves us with the necessity to give denotational semantics to PIPA. In traditional domain theory, this would be a daunting task as even for pure languages, combinations of non-determinism and probabilities require elaborate mathematical techniques [Gou17]. We believe however that our intensional approach gives a new, realistic angle of attack.

This semantics will be obtained by rebuilding the cartesian closed category of concurrent games [CCW19] based on the extension of event structures for combinations of non-determinism and probabilities in **A.1**. We will prove the model computationally adequate, against an operational semantics based on Segala automata, and at first for an angelic resolution of non-determinism (though we can later combine with the developments of [CCHW18] to allow other resolutions of non-determinism). We will aim to prove full abstraction, via a finite definability result.

Adequacy is *medium-term* and *medium-risk*: the required constructions are elaborate, but the way seems clear provided **A.1** is successful. Full abstraction is *high-risk*, but is less critical to the sequel.

[4] It might be counter-intuitive to the reader that we consider PCF$_\parallel$ to be deterministic, since in its usual interleaving operational semantics parallelism would be resolved by non-deterministic scheduling. However, in a *truly concurrent* model non-competing parallel execution remains deterministic. Likewise, we reserve *non-determinism* for external choice for which we can assign no probabilities, so in that sense we do not regard PPCF$_\parallel$ as featuring non-determinism.

B.2 The DyVerSe cube. Building on **B.1**, and following the methodology of Abramsky's *semantic cube* programme [AM99a] we will provide three conditions on strategies, each condition corresponding to the absence of one of the considered computational features. *Innocence* will draw inspiration from our concurrent innocence from [CCW15], and generalize it to this richer setting – innocence corresponds to the absence of side-channel communication. *Probability-determinism* simply consists in forcing all probabilities to be 1, and captures the absence of probabilistic choice. Finally, we will introduce a notion of *sequentiality* (earlier notions seem too restrictive), forbidding parallelism. For each combination of conditions, we will construct a functorial collapse to the matching fully abstract model.

Building on **B.1**, we consider **B.2** as *medium-term* and *low-risk.* The development will take time, due to the variety of other models we have to relate to. If **B.1** only yields adequacy for PIPA, we still regard unifying all other vertices of the DyVerSe to be a worthy achievement.

B.3 Metalanguage and its semantics. Although the DyVerSe cube concerns Call-By-Name languages, it would be a serious flaw for our developments and an obstacle to their adoption if they were inherently limited to Call-By-Name. Objective **B.3** aims to study evaluation order in concurrent games in the light of modern developments in proof theory and the theory of programming languages, and to ensure that our developments on effects do not depend on the evaluation order[5].

The last two decades of research have established the notion of *polarity*, originally proof-theoretical, as a powerful lens through which one may understand atomically issues related to the evaluation order of programs and their computational effects. Milestones include *Polarized Linear Logic*, thoroughly investigated by Laurent [Lau05], Levy's *Call-By-Push-Value* [Lev99], and recent work by Munch-Maccagnoni *et al* [CFM16, MM17] putting together Call-By-Push-Value with the understanding of resources offered by Linear Logic. The latter framework may be regarded as a *metalanguage*, within which high-level features of various programming languages may be understood via translations.

In a sense, games form the setting where the relationship between polarity and evaluation order is the most plain to see. It is reflected by the two interacting players (Player, who plays for the program; and Opponent, who plays for the environment): Call-By-Name languages may be interpreted in *negative games* where Opponent starts, while Call-By-Value languages may be interpreted in *positive games* [HY99] where Player starts. While there is a games model of Call-By-Push-Value [Lev02] following this idea, we are aware of no games model giving a full account of both evaluation order and resources. Reciprocally *concurrency*, whose place appears clearly in concurrent games, is not accounted for by this line of work on polarity and evaluation order. The objectives of **B.3** are two-fold. Firstly, build a concurrent games model of the metalanguage of [CFM16]. This should have the double benefit of exhibiting basic structure in games useful for further model constructions, and of making sure that all the other developments of DyVerSe (and, in particular, those of **B.1** and **B.2**) do not rely on implicit assumptions regarding the evaluation order. Secondly, draw inspiration from this model to design an extension of the metalanguage giving a full status to *concurrent primitives.*

The first objective is *short-term* and *low-risk*, while the second is *medium-term* and more open-ended (but also likely to reach outside of the games community).

Task C. Operation **(P. Clairambault, O. Laurent, Postdoc)**

While game semantics is very successful as a denotational semantics, it is interesting that some of its most striking applications merely exploit its operational content – let us cite applications to hardware compilation [Ghi07] or as a tool for proving decidability of Monadic Second Order logic (MSO) on infinite trees generated by higher-order recursion schemes [Ong06]. In those works game semantics is used to express more locally the complex mechanism of higher-order, possibly effectful computation, as the exchange of "tokens" or the computation of "traversals" in a game-theoretic interaction. Indeed, regarded operationally, game semantics provides techniques to design operational semantics for *open programs*, in such a way that the "results" of the operational semantics may be *composed.* Game semantics, seen through this angle, is *the art of presenting operational semantics compositionally*, or, to quote Ghica, "How to denotational an operational semantics" [Ghi18]. This feature of game semantics is to be emphasized

[5]In game semantics, aspects relative to computational effects are often orthogonal to the evaluation order. For instance, the exact same *innocence* condition corresponds to the absence of shared state with negative games for Call-By-Name [HO00] and for positive games in Call-By-Value [HY99].

when one of the biggest challenge faced by the *formal methods* community is how to design *modular* methods which have a chance to scale, as strongly emphasized by O'Hearn [O'H18]. Hence, it is our conviction that a framework such as the one DyVerSe proposes cannot be complete without a convincing way of connecting concurrent games with operational semantics.

Of course, game semantics is *already* operational in the sense that all the operations used to obtain denotationally the strategy corresponding to a program are effective and can be computed. Nevertheless the inductive reconstruction of a strategy by induction on the program syntax makes it difficult to directly and formally relate actions of the strategy with the program syntax and its operational behaviour as expressed by more standard means. Ideally one wants the best of both worlds: a single semantics that one may regard as being generated either operationally or denotationally, depending on the needs. For traditional game semantics, such connections with operational semantics have been well-studied, starting with [DHR96], connecting Hyland-Ong game semantics with the Krivine Abstract Machine and linear head reduction. Since then researchers have managed to build a much closer connection, essentially presenting game semantics and (open) operational semantics as two sides of the same coin [GT12, LS14, Jab15] – in those works the same object (the strategy) can be regarded as either computed denotationally by induction on syntax, or extracted directly from the operational semantics.

Building such a correspondence in the context of DyVerSe, however, is a challenge. Whereas all operational semantics above are sequential or interleaving-based, our game semantics is truly concurrent. To fulfil our objective we first need an operational semantics fine-grained enough to carry information about causal dependence and independence. The starting point of **Task C** is that a convincing candidate exists under the form of *multi-token Geometry of Interaction* [Lau01], [LFHY14]. Originally introduced by Girard in [Gir89], *Geometry of Interaction (GoI)* provides a methodology to perform higher-order computation via an abstract machine describing the movements of a *token* through the programme syntax[6] – one speaks of a *token machine*. Although GoI does not look like a traditional operational semantics, it certainly deserves the name. Famously, for the pure λ-calculus it was shown to compute Lamping's *optimal reduction* [GAL92], leading to an intense activity around its use for the practical implementation of programming languages, by Mackie [Mac95] among others.

GoI has strong ties with game semantics: Baillot proved in his thesis [Bai99] that for Intuitionistic Multiplicative Exponential Linear Logic (IMELL), GoI could be used to *generate* the strategy interpreting a proof in the sense of game semantics; hence formally linking an operational semantics (the GoI token machine) with a denotational semantics (the game semantics). Although there is, to our knowledge, no other formal result connecting the two, there is little doubt among the experts that the connection remains true for richer languages. Researchers studying the links between games and operational semantics have instead focused on more standard operational settings, such as LTSs.

However, for the purposes of DyVerSe, GoI has a major advantage over more standard approaches to operational semantics: it extends smoothly in the presence of *parallelism*, without resorting to interleavings. For sequential programs, token machines allow only one token through (a graph-based representation of) the program syntax – representing the control flow. However, the local rules describing its movements apply just as well if *several tokens* are present; new mechanisms may then be added to spawn new tokens [Lau01] or synchronize them [LFHY14]. The resulting machines can be then regarded as *colored Petri nets* [P$^+$80]. Now, *unfolding* is one of the most basic and well-studied operations on Petri nets, including colored Petri nets [LHY12]. For regular Petri nets, unfolding yields *event structures* [HW08]. As our concurrent strategies are themselves event structures, it is natural to look for an operational/denotational correspondence as an unfolding.

C.1 Multi-Token GoI and Games for the affine Call-By-Value PCF. Objective **C.1** aims for an *operational/denotational* correspondence by *unfolding*, for affine (*i.e.* no replication) Call-By-Value PCF, with parallel evaluation. We aim for a correspondence along the lines of the diagram below.

$$\begin{tikzcd}
\text{CBV } \mathsf{PCF} \arrow[rr, "\text{Multitoken machine}"] \arrow[dr, "\text{Denotational semantics}"'] & & \text{Colored Petri Net} \arrow[dl, "\text{Unfolding}"] \\
& \text{Concurrent Strategies} &
\end{tikzcd}$$

[6]The focus of [Gir89] was less operational: paired with a realizability construction, GoI was used to build denotational semantics for System F. Over time, the terminology *Geometry of Interaction* has come to focus on the operational aspects.

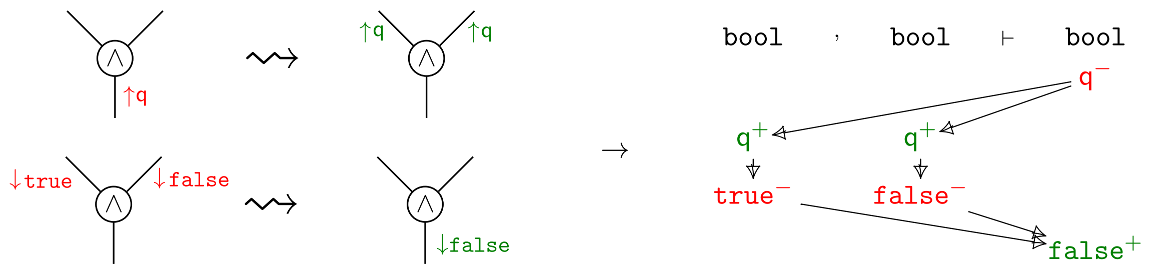

Figure 3: A multi-token machine node and its unfolding as a concurrent strategy

We illustrate the idea in Figure 3 (though in Call-By-Name, with a parallel evaluation of the boolean *and* $\wedge$). A program is represented graphically via combining *nodes*, linked by *edges*. The left hand side of Figure 3 represents a (rather trivial) token machine, with only node the boolean operation $\wedge$. Edges may carry *tokens* following a direction, and each node has an action on incoming tokens specified by local *transitions*. Here we show two of the transitions of $\wedge$: the top one expresses that upon an evaluation request, $\wedge$ starts the evaluation of its two arguments in parallel by sending two tokens. The bottom one expresses that upon receiving `true` and `false`, $\wedge$ synchronises them and replies `false`. Dangling edges have *external transitions* (not shown in Figure 3) induced by the type represented as a *game*. *Unfolding*, in the same technical sense as for Petri nets, yields the diagram on the right hand side, with Opponent moves in red (from the environment), Player moves in green (from the program), and the Hasse diagram of their causal dependencies: a fragment of a concurrent strategy.

We feel like this objective is a good place to start, laying the basis for further work: there is already a multi-token GoI machine for full Call-By-Value PCF with parallel evaluation (meaning that in $M\,N$, the function and the argument are evaluated in parallel) [LFHY14]. Likewise, the corresponding games model is already well-understood; it is the linear fragment of [CdVW19] (ignoring quantum primitives). The absence of replication simplifies the games model dramatically, as there is no need for symmetry [CCW14]. Hence this should serve as a proof of concept, and to set up the proof techniques. Once this basic picture is in place, we will extend it to the affine fragment of the metalanguage of **B.3**.

We consider this as *medium-term* and *medium-risk*; we foresee no particular difficulty, but to our knowledge nothing of the sort has been done before, and the proof methods have to be developed.

C.2 Multi-Token GoI and Games for IPA. Once **C.1** has established the basic principles of our methodology relating GoI to concurrent games via unfoldings, the next step is to deal with replication on the one hand, and shared state on the other hand. For these we will devise a multi-token machine, drawing inspiration from the *memoryfull GoI* of [MHH16]. On the games side we will aim to unfold to the truly concurrent games of IPA introduced in [CCW19]. To our knowledge, this will be the first GoI machine for a concurrent programming language with shared state.

We consider this as *medium-term* and *medium-risk*. Hopefully we can rely on the proof techniques of **C.1** which will have be developed with this extension in mind.

C.3 GoI and Unfoldings for the DyVerSe cube. Objective **C.3** aims to extend **C.2** to represent all the primitives of PIPA. For that, we will extend the probabilistic multi-token machine of [LFVY17] to deal with non-deterministic choice as well, and construct the unfolding to the strategies of **B.1**.

We consider **C.3** as *long-term* and *high-risk*. There is significant prior research to do; for instance we will have to do some preliminary work on unfoldings of probabilistic extensions of Petri nets. Though objective **C.3** is very desirable to tie **Track B** and **Track C** together, we regard it as less fundamental than **C.2** and **C.4** for the success and impact of DyVerSe.

C.4 Implementation. Objective **C.4** aims at implementing the constructions and results of **Task C**. While there are already implementations of GoI, no such implementation exists, to our knowledge, for multi-token machines. Besides, while these implementations focus on actually *executing* GoI, our implementation will focus on computing *unfoldings* and presenting them as strategies. Via these unfoldings, our implementation will be able to directly present to the user a representation of the concurrent strategy corresponding to a program, and allow them to interactively play with it.

We regard this implementation as a crucial outcome of DyVerSe, hence we will start it as soon as possible after **C.1** is completed and expand it as **Task C** develops. The main purpose of the implementation is to provide a platform with which one can *visualize* and *experiment* with the constructions of DyVerSe; which we believe is essential in the diffusion and impact of our developments and ideas. Nevertheless, once the implementation is developed, we will investigate its applicability as a tool in software development: in particular, we have in mind interactive debugging for higher-order effectful languages – some of the applications of **Task E** also rely on and extend the implementation.

That we manage to develop the implementation is *low-risk* and *medium-term*: while we lack an established track record in software development, we are confident that we have enough experience in programming (notably in Ocaml) for this. Nevertheless, the quality and scope of the implementation would certainly benefit from additional experience. Therefore, prior programming experience will be an important selection criterion for the DyVerSe postdoc, who will contribute to **C.4**.

Task D. Interference (P. Clairambault, G. Munch-Maccagnoni, C. Riba, PhD student)

The developments of **Tasks B and C** aim to capture the execution of complex programming languages "in the wild", representing faithfully their potential untamed, unproductive behaviours such as *deadlocks* or *data races* – the latter is an example of what Reynolds calls *interference*[7] [Rey78]. In contrast, many of the developments of programming language theory aim to design structures that constrain the execution, ensuring by construction or by verification that the behaviour of programs is *defined*, *i.e.* that the programs are deterministic up to the choice of the scheduler. As an early example of that, Reynold's *Syntactic Control of Interference* [Rey78] (with later improvements by O'Hearn *et al* [OPTT99]) forbids unwanted interference by forcing distinct active (potentially mutated) identifiers to refer to distinct locations in memory. Syntactic Control of Interference (SCI) is amongst the inspirations of Bunched Implications [OP99] which stands behind Separation Logic [O'H19], one of the most striking contributions of logic to formal methods of the past decades. Finally, Mozilla's *Rust* [JJKD18] makes the practical demonstration of these ideas at scale: Rust forbids *data races* with a notion of *ownership* determined by an *affine* typing rule familiar from Linear Logic, to which it adds an elaborate *borrowing* mechanism reminiscent of SCI, so as to increase expressivity.

As such constructs have an increasing importance in modern programming languages, **Task D** aims to answer the need for semantic tools to prove their soundness, and study in a principled way the design space around them. In [CCW15], we have exploited true concurrency to present deterministically the parallel execution of the pure higher-order programming language PCF, hence giving a concrete, geometric representation of the fact it is interference-free. We regard this result as a proof-of-concept for **Task D**. We aim to extend it, and exploit the corresponding models to reason about interference.

D.1 Game Semantics for SCI. We believe that a satisfactory semantic understanding of interference should start with SCI; because it is simpler, and because earlier investigations on its semantics provide a starting point. In particular, in his PhD [Wal04], Wall has constructed a game semantics for *sequential* SCI – interestingly, his model involved replacing the *view* familiar from traditional game semantics with a partially ordered version, similar to structures from our concurrent games.

Objective **D.1** will start by investigating *Basic SCI*. Basic SCI [OPTT99] is a Call-By-Name concurrent programming language with shared state, with typing rules restricted in the following way.

$$\frac{\Gamma \vdash M : A \to B \qquad \Delta \vdash N : A}{\Gamma, \Delta \vdash M\,N : B} \qquad \frac{\Gamma \vdash M : \mathbf{com} \qquad \Delta \vdash N : \mathbf{com}}{\Gamma, \Delta \vdash M \parallel N : \mathbf{com}} \qquad \frac{\Gamma \vdash M : \mathbf{com} \qquad \Gamma \vdash N : \mathbf{com}}{\Gamma \vdash M;\, N : \mathbf{com}}$$

where Γ, Δ implies that the two are disjoint (and **com** is the type of *commands* that may perform some side effects before possibly evaluating to a single value **skip**). The typing rules imply that identifiers may not be shared when spread over an application or parallel composition. However, they may be shared over a sequential composition, where it is clear that no data race can occur. We will first build a concurrent game semantics for Basic SCI, where programs are interpreted by *deterministic strategies*. Then this will

[7] This use of *interference* is not related to its widespread use in the context of security of information flow [RS01].

be extended to SCIR [OPTT99], which extends Basic SCI with a notion of passive types that may be shared at will. In other words, **D.1** will yield a concurrent version of [Wal04].

We regard this as *medium-term* and *low-risk* – we expect to be able to take significant inspiration from [Wal04]. This step is however necessary, drawing useful lessons for further research.

D.2 Games Semantics for Idealized Rust. Rust is a recent programming language due to Mozilla Research, which provides the programmer both a low-level control over resource management, and strong guarantees about safety and the absence of data races. For that, it relies on an ownership-and-borrowing design not unlike SCI, but much more elaborate: there are in particular means to *transfer* ownership of mutable variables over channels, and *share* read-only or mutable references in a suitably controlled way. Rust itself does not have a formal description or semantics; but Dreyer's ERC project *RustBelt* aims to provide Rust with theoretical foundations – in particular, in their striking recent work [JJKD18], the RustBelt team gave a sound and adequate logical relation semantics (formalized in Iris) for a formalized fragment λ_{Rust} of Rust, yielding a soundness proof for the type system. Also, the calculus *Oxide* presented very recently [WPMA19] is complementary to RustBelt and gives a faithful account of the ownership-and-borrowing mechanism.

We argue that ownership is causal in essence. Objective **D.2** aims to provide a causal analysis of the fundamental mechanisms behind the design of Rust. This will be done by constructing a concurrent games model of a language *Idealized Rust* featuring ownership transfer and shared borrowing (this language must be defined, but good inspirations exist in λ_{Rust} or Oxide). This development will build on **D.1**, but also on **A.3** as we expect that treating ownership will require the use of *names* as identifiers of owned resources. As an application of our model, we will investigate how the structures interact with evaluation order as specified by the metalanguage of **B.3**, and relate it with our recent algebraic model of the resource-management contents of ownership in Rust [CM18]. In particular, we expect this analysis to be be fruitful in our ongoing proposal to port Rust's model to functional programming languages, in particular to the upcoming multicore OCaml [Mun18].

We regard this as being *medium-term* and *medium-risk.* Though soundness of this fragment of Rust will follow and made more robust and compositional, the ambition of DyVerSe is not to compete with Oxide or RustBelt. Rather, we aim to put the structures behind Rust under a different light, informing new designs as with our OCaml proposal, or being more amenable to automation than logical relations.

D.3 Strategies as Safe Petri Nets. Although not designed primarily for this purpose, we believe that the interference-free aspect of a language such as Basic SCI enables an effective representation of its concurrent game semantics as investigated in **D.1**, amenable to algorithmic treatment.

In **Task C** we develop, through GoI, an interpretation of programs as certain colored Petri nets. This interpretation is badly infinitary: the colors themselves come from an infinite alphabet, as per their use for copy indexing in a language with unrestricted replication of resources. Yet, the case of Basic SCI is special: replication is allowed, but only sequentially. In other words resources must be "freed" before being reused. This imposed sequential ordering on resource usage means that copy indexing may be done implicitly via this ordering, rather than through explicit indices as in GoI, so that one may get rid of the colors of tokens. Furthermore, because of the non-interference property of Basic SCI, we believe that only one token may occupy a place at a given time, so that the GoI version of our games may be formulated as a *(1)-safe* Petri net. This is promising, because safe Petri nets are one of the few classes of Petri nets with good decidability properties, such as trace equivalence or even the truly concurrent *pomset equivalence* [EN94]. We are aware that Basic SCI is a very poor language in terms of expressivity: our positioning in **D.3** is more to identify decidable fragments for the very hard problems of verifying higher-order stateful concurrent programs, rather than providing a practical verification tool. In that our main inspiration is the celebrated result of Ghica and McCusker [GM03] presenting strategies for second-order Idealized Algol as finite automata, and the subsequent line of work of *algorithmic game semantics* [AGMO04].

Relying on objectives **C.2** and **D.1**, objective **D.3** aims to present the concurrent strategies for Basic SCI (with finite datatypes) developed in objective **D.1** as unfoldings of safe Petri nets. We will also investigate the possibilities this might offer for automated verification. There are also strong links between the proposed interpretation of Basic SCI as Petri nets and Ghica's Geometry of Synthesis [Ghi07] which also relies on a language like SCI for hardware compilation. We will henceforth attempt to apply our model to give a truly concurrent soundness proof of Ghica's hardware compilation procedure.

We regard the core objective of **D.3** as *short-term* and *medium-risk* in the sense that provided its dependencies are met, it shall be easily seen whether an interpretation of programs as safe Petri nets as envisaged is possible. Applications of this construction, including the relationship with Geometry of Synthesis, are open-ended and therefore more long-term.

Task E. Applications (P. Clairambault, C. Riba)

The core contribution of DyVerSe is a theoretical framework, which we hope will allow a sharper understanding of causal structures behind high-level and effectful concurrent programming languages. However, we believe the sign of a useful theoretical framework is that it provides tools relevant to other communities of researchers. Both for the scientific well-foundedness of our investigations and to ensure the visibility of our work, we will develop *applications* of our tools outside of game semantics.

Causality is a fundamental notion, relevant to many applications ranging from weak memory concurrency [Alg12] to security [CW01] or software diagnosis [BFHJ03]. We believe there is tremendous untapped potential in the causal analysis of complex programs and systems offered by DyVerSe and concurrent games in general. Concurrent games have already been put to work outside of game semantics, with recent contributions highlighting causal structures in logic [ACHW18], [CY19], solving an open problem in probabilistic coherence spaces [CP18], or supporting Bayesian inference in probabilistic programming [CP19], but we have barely scratched the surface. Accordingly the objectives below are far from exhaustive regarding the potential applications of concurrent games. They have been selected for their potential impact, and their synergy with the rest of DyVerSe.

E.1 Parallel Higher-Order Model-Checking. *Higher-Order Model-Checking* (HOMC) proposes an approach to the automated verification of higher-order programs. This line of work, actively developed in the past 15 years, was prompted by Ong's celebrated result that *Monadic Second-Order Logic (MSO)*, a powerful specification logic, is decidable on infinite trees generated by Higher-Order Recursion Schemes [Ong06]. Though the complexity is in principle prohibitive (non-elementary), Kobayashi (among others) demonstrated that HOMC, coupled with predicate abstraction and Counter-Example Guided Abstraction Refinement (CEGAR) could be used in practice to verify higher-order programs [Kob09, KSU11], and reported surprisingly good performance [Kob13].

HOMC mostly concerns sequential higher-order programs and it might seem that an extension to concurrent programs is hopeless; after all, even the pairwise reachability analysis of two synchronizing pushdown systems is undecidable [Ram00]. Despite this fundamental limitation, Yasukata *et al* proved that pairwise reachability was decidable for concurrent higher-order programs if the synchronization mechanism is restricted to *nested locks* [YKM14]. Interestingly, this is done by reduction to HOMC, translating the program to a non-deterministic scheme implementing its interleaving semantics, and using that the language of branches respecting the lock discipline is regular. Seen from the perspective of DyVerSe this invites the question: what properties could we express and verify if, rather than an interleaving semantics, we had direct access to the causal, truly concurrent representation of the execution of such programs? Could we decide properties of behavioural logics designed to express truly concurrent properties [BC14] such as causality or independence?

For sequential programs, the execution traces of a program are obtained as the *Böhm tree* of the scheme representing the program, *i.e.* its infinite normal form. But how may we proceed if, as in [YKM14], the execution is no longer a tree, but a labeled partial order? Interestingly, in Ong's original proof [Ong06], the Böhm tree was described as the game semantical *strategy* of the scheme to be analysed, which is isomorphic to the Böhm tree. While Böhm trees do not apply in this truly concurrent scenario, our concurrent strategies do, forming the basis of our proposed investigation.

In objective **E.1**, we propose to model-check *concurrent strategies* corresponding to higher-order concurrent programs with restricted synchronization primitives. For that we will work on the metalanguage of **B.3** (so as to insure, as in [CGM18], that we cover directly all evaluation orders), extend it with nested locks and concurrent game semantics. Drawing inspiration from our earlier work on HOMC [CGM18] we will design a decidable intersection type system for the language, and prove its soundness by relating it to the concurrent game semantics (in the spirit of [TO14]). Besides we will investigate the expressivity of the intersection type system on the generated concurrent strategies, in terms of fragments of the logic for true concurrency of [BC14].

We regard this task as *medium-term* and *medium-risk*.

E.2 Verification by Symbolic Execution. Verification of concurrent programs is notoriously difficult, because exact methods bump against a very low undecidability frontier, and both exact and approximate methods struggle with the so-called *state explosion problem* caused by the high number of possible interleavings. Notable approaches to avoid the state explosion problem include *partial order reduction* [FG05] and *net unfoldings* [Esp94]; both exploit the independence of certain sub-computations in order to reduce the search space. Though a wealth of extensions has been considered, to our knowledge such methods have not been applied to high-level concurrent programming languages such as those considered in DyVerSe, with features such as higher-order functions. One significant difficulty in such an endeavor is to extract from higher-order effectful programs a first-order model amenable to automated verification. Game semantics was quickly identified as a relevant tool to perform this extraction [GM06], for higher-order sequential programs – first for finite-state verification, more recently extended by infinite state verification through symbolic computation [Dim14].

In **E.2** we propose to investigate the efficiency of the representation of programs offered by DyVerSe for verification purposes. In contrast to **D.3** and **E.1** which aim to identify decidable complete verification problems, in **E.2** we attack infinite state programs through symbolic computation, and are perfectly content with approximate methods. We will enrich the implementation of **C.4** with symbolic computation, relying on the theoretical foundations established in **A.2**. We will experiment with applying this to verification and testing, for instance following the symbolic computation of Petri nets described in [GMMP89]. Validation will be experimental, through the development of benchmarks.

We regard **E.2** as *medium-term* and *high-risk*. Though it could have notable impact, none of us are experts in automated program verification nor have significant experience with experimental validation.

E.3 Witness extraction in arithmetical theories. The theory of programming languages has had, in the past 40 years, deep and fruitful interactions and cross-fertilization with mathematical logic and *proof theory* in particular, alongside the *Curry-Howard correspondence* [SU06] between proofs and programs. Following this tradition, we are also keen to develop applications of our causal methods in logic; and in particular to *proof mining*. In fact we have already done so: in [ACHW18] we have constructed a concurrent games model for first-order classical proofs, which allows us to extract the computational content of proof in the sense of Herbrand disjunctions [GK05]. In that work, we represent as event structures the causality between quantifiers inherent in a classical proof, and terms in the Herbrand disjunction are extracted by composing partial witnesses alongside the causal path. Our model is original, in that it avoids the use of higher-order functions that is usually associated with witness extraction, as for instance with Gödel's *Dialectica interpretation* [AF98].

In **E.3** we propose to extend the methodology of [ACHW18] beyond pure first-order proofs, in the context of theories of arithmetic. Relying on the structures of **A.2**, we will build a concurrent games model of Peano Arithmetic (PA) with events annotated with arithmetical functions (notably, the ability to remove inconsistent events is crucial to deal with equality, to discard wrong equality claims). We will first reprove – as a proof of concept application – the classical characterisation of provably total functions in PA with induction restricted to Σ_n formulas. As for pure first-order proofs our method will avoid the use of higher-order functionals central to techniques such as Gödel's Dialectica [AF98] or realizability [Kri09]. To our knowledge, avoiding higher-order functionals in witness extraction is for now a specificity of the *witness function method* [Bus95] (used in particular in the context of theories of bounded arithmetic where functional interpretations do not apply). We will seek formal links with the witness function methods, uncovering semantic structures in proof mining.

We regard **E.3** as a *medium-term* and *medium-risk* objective. Though we are confident in obtaining a model of PA, its usability in proof mining and relationship to existing tools are rather open-ended.

E.4 Proof Methods for Effectful Probabilistic Program Equivalence. There are a number of well-established models and techniques to reason about equivalences between probabilistic processes [DEP02, vBMOW05], or associated notions of metrics [DGJP04]. For higher-level probabilistic languages

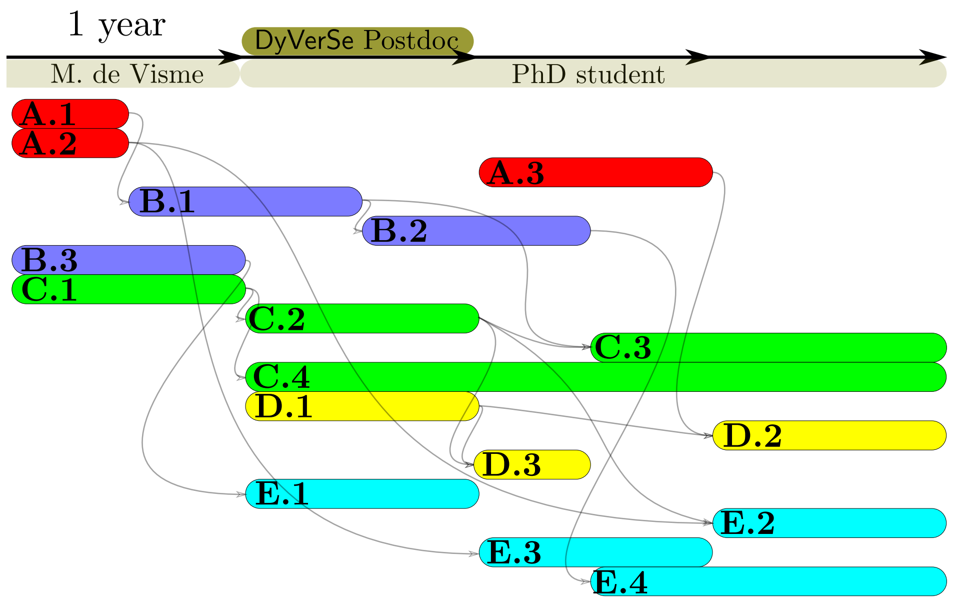

Figure 4: Organization in time and dependencies between tasks

such as extensions of the λ-calculus, notions of program equivalence or techniques to establish it have been actively investigated in recent years. Developments include probabilistic notions of applicative bisimulation [CL14] or logical relations [BB15]; quantitative extensions to *metrics* have also been considered [CL17]. These developments however are still at an early stage, and to our knowledge do not scale to languages comparing in expressivity with *e.g.* PIPA. Game semantics, with its wealth of full abstraction results, is well-known in the classical setting as a powerful tool to reason about program equivalence; we want to exploit this in the probabilistic case.

Objective **E.4** aims to leverage and extend the semantic developments of **Task B**, and in particular objective **B.2**, to study notions of equivalence for complex probabilistic programs, including concurrency and shared state. We will first characterise concretely observational equivalence for PIPA, and study its effective presentability. We will give a general study of the impact of adding probabilistic choice to programming languages of the DyVerSe cube in terms of observational equivalence. We will design semantic structures in concurrent games generalizing the metric study of [CL17], and derive proof techniques to bound distance between complex probabilistic programs.

We regard **E.4** as *long-term* and *high-risk*. If **B.2** is successful, we are confident of its impact as to the study of equivalences between probabilistic programs. The extension to metrics is very open-ended.

Organization in time and risk management

Figure 4 illustrates the organization of DyVerSe in time. We indicate the envisaged presence of non-permanent researchers, emphasizing the postdoc funded by the project. We also indicate the dependencies between tasks. The diagram is of course indicative, and incomplete: for instance, objective **B.3**, although having few strict dependencies, will have if successful an impact on most of the other tasks.

In the description of the individual tasks, we have attempted to point out what were the main risks or areas of uncertainties, and which fallbacks mitigated them. In designing DyVerSe we kept in mind the desire that should any objective fail completely, the rest of the project will retain sufficient integrity for DyVerSe to have significant impact.

2 Impact and Benefits

We will publish our results in top journals (*Logical Methods in Computer Science*, *Mathematical Structures in Computer Science*, *Theoretical Computer Science*, *Transactions on Programming Languages and Systems*) and conferences (LICS, POPL, CONCUR, FoSSaCS, ICALP, FSCD…) relevant for theory of programming languages, with an eye on open access. Our track record witnesses our ability to do so.

The implementation of **C.4** will be open-source, available on GitHub, and we will submit an associated paper to the international conference TACAS. We will also make a demo available online.

Some objectives of DyVerSe (most notably, **E.2**) may impact in the medium-term the practice of automated verification of programs. We will, of course, investigate the possibility and invest effort on it, should it arise. Most notably, the scientific methodology of DyVerSe is resolutely long-term, and aims at progress in our understanding of the fundamental structures behind rich programming languages, so as to better design them, reason on them and implement them. We believe that the contributions of DyVerSe, if properly disseminated towards the scientific community, have the potential to become part of the essential toolbox in theory of programming languages. Historically, the wide diffusion of game semantic tools has been hampered by its technicality, an obstacle to understanding. This lead to an unfortunate situation, with researchers from other communities reinventing limited forms of game semantics according to their needs, without necessarily being aware that relevant mathematical tools were available (see for instance [DF15, SBCA15]). While not an end in itself, we believe our implementation can allow researchers to identify the results of DyVerSe by seeing them at play, and help give our fundamental results visibility and a long-lasting impact on the scientific community.

References to our publications

[ACHW18] Aurore Alcolei, Pierre Clairambault, Martin Hyland, and Glynn Winskel. The true concurrency of Herbrand's theorem. In CSL, 2018.

[ACL19] Aurore Alcolei, Pierre Clairambault, and Olivier Laurent. Resource-tracking concurrent games. In FoSSaCs, 2019.

[CCHW18] Simon Castellan, Pierre Clairambault, Jonathan Hayman, and Glynn Winskel. Non-angelic concurrent game semantics. In FoSSaCS, 2018.

[CCPW18] Simon Castellan, Pierre Clairambault, Hugo Paquet, and Glynn Winskel. The concurrent game semantics of probabilistic PCF. In LICS, 2018.

[CCW14] Simon Castellan, Pierre Clairambault, and Glynn Winskel. Symmetry in concurrent games. In CSL-LICS, 2014.

[CCW15] Simon Castellan, Pierre Clairambault, and Glynn Winskel. The parallel intensionally fully abstract games model of PCF. In LICS, 2015.

[CCW19] Simon Castellan, Pierre Clairambault, and Glynn Winskel. Thin games with symmetry and concurrent Hyland-Ong games. Logical Methods in Computer Science, 2019.

[CdVW19] Pierre Clairambault, Marc de Visme, and Glynn Winskel. Game semantics for quantum programming. PACMPL, (POPL), 2019.

[CFM16] Pierre-Louis Curien, Marcelo P. Fiore, and Guillaume Munch-Maccagnoni. A theory of effects and resources: adjunction models and polarised calculi. In POPL, 2016.

[CGM18] Pierre Clairambault, Charles Grellois, and Andrzej S. Murawski. Linearity in higher-order recursion schemes. PACMPL, (POPL), 2018.

[CGW12] Pierre Clairambault, Julian Gutierrez, and Glynn Winskel. The winning ways of concurrent games. In LICS, 2012.

[CM18] Guillaume Combette and Guillaume Munch-Maccagnoni. A resource modality for RAII. In LOLA, 2018.

[CP18] Pierre Clairambault and Hugo Paquet. Fully abstract models of the probabilistic lambda-calculus. In CSL, 2018.

[Lau01] Olivier Laurent. A token machine for full geometry of interaction. In TLCA, 2001.

[Lau05] Olivier Laurent. Syntax vs. semantics: A polarized approach. Theor. Comput. Sci., 2005.

[MM17] Guillaume Munch-Maccagnoni. Note on Curry's style for Linear Call-by-Push-Value. In preparation, 51 pages. hal-01528857, 2017.

[Mun18] Guillaume Munch-Maccagnoni. Resource polymorphism. CoRR, abs/1803.02796, 2018.

References to others

[AF98] Jeremy Avigad and Solomon Feferman. Gödel's functional ("Dialectica") interpretation. Handbook of proof theory, 1998.

[AGMO04] Samson Abramsky, Dan R. Ghica, Andrzej S. Murawski, and C.-H. Luke Ong. Applying game semantics to compositional software modeling and verification. In TACAS, 2004.

[AJM00] Samson Abramsky, Radha Jagadeesan, and Pasquale Malacaria. Full abstraction for PCF. Inf. Comput., 2000.

[Alg12] Jade Alglave. A formal hierarchy of weak memory models. Formal Methods in System Design, 2012.

[AM96] Samson Abramsky and Guy McCusker. Linearity, sharing and state: a fully abstract game semantics for Idealized Algol with active expressions. Electr. Notes Theor. Comput. Sci., 1996.

[AM99a] Samson Abramsky and Guy McCusker. Game semantics. In Computational logic, 1999.

[AM99b] Samson Abramsky and Paul-André Melliès. Concurrent games and full completeness. In LICS, 1999.

[Arm10] Joe Armstrong. Erlang. Commun. ACM, 2010.

[Bai99] Patrick Baillot. Approches dynamiques en sémantique de la logique linéaire: jeux et géométrie de l'interaction. PhD thesis, Aix-Marseille 2, 1999.

[BB15] Ales Bizjak and Lars Birkedal. Step-indexed logical relations for probability. In FoSSaCS, 2015.

[BC14] Paolo Baldan and Silvia Crafa. A logic for true concurrency. J. ACM, 2014.

[BFHJ03] Albert Benveniste, Eric Fabre, Stefan Haar, and Claude Jard. Diagnosis of asynchronous discrete-event systems: a net unfolding approach. IEEE Trans. Automat. Contr., 2003.

[Bus95] Samuel R Buss. The witness function method and provably recursive functions of peano arithmetic. In Studies in Logic and the Foundations of Mathematics. 1995.

[CL14] Raphaëlle Crubillé and Ugo Dal Lago. On probabilistic applicative bisimulation and call-by-value lambda-calculi. In ESOP, 2014.

[CL17] Raphaëlle Crubillé and Ugo Dal Lago. Metric reasoning about lambda-terms: The general case. In ESOP, 2017.

[CP19] Simon Castellan and Hugo Paquet. Probabilistic programming inference via intensional semantics. In ESOP, 2019.

[CW01] Federico Crazzolara and Glynn Winskel. Events in security protocols. In CCS, 2001.

[CY19] Simon Castellan and Nobuko Yoshida. Causality in linear logic: Full completeness and injectivity (unit-free multiplicative-additive fragment). In FoSSaCS, 2019.

[DD07] Yuxin Deng and Wenjie Du. Probabilistic barbed congruence. Electr. Notes Theor. Comput. Sci., 2007.

[DEP02] Josee Desharnais, Abbas Edalat, and Prakash Panangaden. Bisimulation for labelled markov processes. Inf. Comput., 2002.

[DF15] Tim Disney and Cormac Flanagan. Game semantics for type soundness. In LICS, 2015.

[DGJP04] Josee Desharnais, Vineet Gupta, Radha Jagadeesan, and Prakash Panangaden. Metrics for labelled markov processes. Theor. Comput. Sci., 2004.

[DH00] Vincent Danos and Russell Harmer. Probabilistic game semantics. In LICS, 2000.

[DHR96] Vincent Danos, Hugo Herbelin, and Laurent Regnier. Game semantics & abstract machines. In LICS, 1996.

[Dim14] Aleksandar S. Dimovski. Program verification using symbolic game semantics. Theor. Comput. Sci., 2014.

[EN94] Javier Esparza and Mogens Nielsen. Decidability issues for petri nets - a survey. Bulletin of the EATCS, 1994.

[EPT18] Thomas Ehrhard, Michele Pagani, and Christine Tasson. Full abstraction for probabilistic PCF. J. ACM, 2018.

[Esp94] Javier Esparza. Model checking using net unfoldings. Sci. Comput. Program., 1994.

[FG05] Cormac Flanagan and Patrice Godefroid. Dynamic partial-order reduction for model checking software. In POPL, 2005.

[FP09] Claudia Faggian and Mauro Piccolo. Partial orders, event structures and linear strategies. In TLCA, 2009.

[GAL92] Georges Gonthier, Martín Abadi, and Jean-Jacques Lévy. The geometry of optimal lambda reduction. In POPL, 1992.

[Ghi07] Dan R. Ghica. Geometry of synthesis: a structured approach to VLSI design. In POPL, 2007.

[Ghi18] Dan Ghica. How to denotational an operational semantics, 2018. Talk at *Game Semantics 25*, satellite workshop of LICS.

[Gir87] Jean-Yves Girard. Linear logic. Theor. Comput. Sci., 1987.

[Gir89] Jean-Yves Girard. Geometry of interaction 1: Interpretation of system F. In Studies in Logic and the Foundations of Mathematics. 1989.

[GK05] Philipp Gerhardy and Ulrich Kohlenbach. Extracting herbrand disjunctions by functional interpretation. Arch. Math. Log., 2005.

[GM03] Dan R. Ghica and Guy McCusker. The regular-language semantics of second-order Idealized Algol. Theor. Comput. Sci., 2003.

[GM06] Dan R. Ghica and Andrzej S. Murawski. Compositional model extraction for higher-order concurrent programs. In TACAS, 2006.

[GM08] Dan R. Ghica and Andrzej S. Murawski. Angelic semantics of fine-grained concurrency. Ann. Pure Appl. Logic, 2008.

[GMMP89] Carlo Ghezzi, Dino Mandrioli, Sandro Morasca, and Mauro Pezzè. Symbolic execution of concurrent systems using petri nets. Comput. Lang., 14(4):263–281, 1989.

[Gor00] Andrew D. Gordon. Notes on nominal calculi for security and mobility. In FOSAD, volume 2171 of Lecture Notes in Computer Science, pages 262–330. Springer, 2000.

[Gou17] Jean Goubault-Larrecq. Isomorphism theorems between models of mixed choice. Mathematical Structures in Computer Science, 2017.

[GT12] Dan R. Ghica and Nikos Tzevelekos. A system-level game semantics. Electr. Notes Theor. Comput. Sci., 2012.

[Har99] Russell Harmer. Games and full abstraction for non-deterministic languages. PhD thesis, Imperial College London, UK, 1999.

[HM99] Russell Harmer and Guy McCusker. A fully abstract game semantics for finite nondeterminism. In LICS, 1999.

[HO00] J. M. E. Hyland and C.-H. Luke Ong. On full abstraction for PCF: I, II, and III. Inf. Comput., 2000.

[HW08] Jonathan Hayman and Glynn Winskel. The unfolding of general petri nets. In FSTTCS, 2008.

[HY99] Kohei Honda and Nobuko Yoshida. Game-theoretic analysis of call-by-value computation. Theor. Comput. Sci., 1999.

[Jab15] Guilhem Jaber. Operational nominal game semantics. In FoSSaCS, 2015.

[JJKD18] Ralf Jung, Jacques-Henri Jourdan, Robbert Krebbers, and Derek Dreyer. Rustbelt: securing the foundations of the rust programming language. In POPL, 2018.

[Kob09] Naoki Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. In POPL, 2009.

[Kob13] Naoki Kobayashi. Model checking higher-order programs. J. ACM, 2013.

[Kri09] Jean-Louis Krivine. Realizability in classical logic. Panoramas et synthèses, 2009.

[KSU11] Naoki Kobayashi, Ryosuke Sato, and Hiroshi Unno. Predicate abstraction and CEGAR for higher-order model checking. In PLDI, 2011.

[Lai97] James Laird. Full abstraction for functional languages with control. In LICS, 1997.

[Lai01] James Laird. A fully abstract game semantics of local exceptions. In LICS, 2001.

[Ler09] Xavier Leroy. Formal verification of a realistic compiler. Commun. ACM, 2009.

[Lev99] Paul Blain Levy. Call-by-push-value: A subsuming paradigm. In TLCA, 1999.

[Lev02] Paul Blain Levy. Adjunction models for call-by-push-value with stacks. Electr. Notes Theor. Comput. Sci., 2002.

[LFHY14] Ugo Dal Lago, Claudia Faggian, Ichiro Hasuo, and Akira Yoshimizu. The geometry of synchronization. In CSL-LICS, 2014.

[LFVY17] Ugo Dal Lago, Claudia Faggian, Benoît Valiron, and Akira Yoshimizu. The geometry of parallelism: classical, probabilistic, and quantum effects. In POPL, 2017.

[LHY12] Fei Liu, Monika Heiner, and Ming Yang. An efficient method for unfolding colored petri nets. In WSC, 2012.

[LS14] Paul Blain Levy and Sam Staton. Transition systems over games. In CSL-LICS, 2014.

[Mac95] Ian Mackie. The geometry of interaction machine. In POPL, 1995.

[Mel04] Paul-André Melliès. Asynchronous games 2: The true concurrency of innocence. In CONCUR, 2004.

[MHH16] Koko Muroya, Naohiko Hoshino, and Ichiro Hasuo. Memoryful geometry of interaction II: recursion and adequacy. In POPL, 2016.

[MM07] Paul-André Melliès and Samuel Mimram. Asynchronous games: Innocence without alternation. In CONCUR, 2007.

[MT11] Andrzej S. Murawski and Nikos Tzevelekos. Game semantics for good general references. In LICS, 2011.

[MT16] Andrzej S. Murawski and Nikos Tzevelekos. Nominal game semantics. Foundations and Trends in Programming Languages, 2016.

[NPW79] Mogens Nielsen, Gordon D. Plotkin, and Glynn Winskel. Petri nets, event structures and domains. In SCC, 1979.

[O'H18] Peter W. O'Hearn. Continuous reasoning: Scaling the impact of formal methods. In LICS, 2018.

[O'H19] Peter W. O'Hearn. Separation logic. Commun. ACM, 2019.

[Ong06] C.-H. Luke Ong. On model-checking trees generated by higher-order recursion schemes. In LICS, 2006.

[OP99] Peter W. O'Hearn and David J. Pym. The logic of bunched implications. Bulletin of Symbolic Logic, 1999.

[OPTT99] Peter W. O'Hearn, John Power, Makoto Takeyama, and Robert D. Tennent. Syntactic control of interference revisited. Theor. Comput. Sci., 1999.

[P$^+$80] James L Peterson et al. A note on colored petri nets. Inf. Process. Lett., 1980.

[Pit13] Andrew M Pitts. Nominal Sets: Names and Symmetry in Computer Science. Cambridge University Press, 2013.

[Plo77] Gordon D. Plotkin. LCF considered as a programming language. Theor. Comput. Sci., 1977.

[PW18] Hugo Paquet and Glynn Winskel. Continuous probability distributions in concurrent games. Electr. Notes Theor. Comput. Sci., 2018.

[Ram00] G. Ramalingam. Context-sensitive synchronization-sensitive analysis is undecidable. ACM Trans. Program. Lang. Syst., 2000.

[Rey78] John C. Reynolds. Syntactic control of interference. In POPL, 1978.

[RS01] Peter Y. A. Ryan and Steve A. Schneider. Process algebra and non-interference. Journal of Computer Security, 2001.

[RW11] Silvain Rideau and Glynn Winskel. Concurrent strategies. In LICS, 2011.

[SBCA15] Gordon Stewart, Lennart Beringer, Santiago Cuellar, and Andrew W. Appel. Compositional compcert. In POPL, 2015.

[Seg95] Roberto Segala. Modeling and verification of randomized distributed real-time systems. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 1995.

[Seg06] Luc Segoufin. Automata and logics for words and trees over an infinite alphabet. In CSL, 2006.

[Sto02] Mariëlle Stoelinga. An introduction to probabilistic automata. Bulletin of the EATCS, 2002.

[SU06] Morten Heine Sørensen and Pawel Urzyczyn. Lectures on the Curry-Howard isomorphism, volume 149. Elsevier, 2006.

[SYW$^+$16] Sam Staton, Hongseok Yang, Frank D. Wood, Chris Heunen, and Ohad Kammar. Semantics for probabilistic programming: higher-order functions, continuous distributions, and soft constraints. In LICS, 2016.

[TO14] Takeshi Tsukada and C.-H. Luke Ong. Compositional higher-order model checking via omega-regular games over böhm trees. In CSL-LICS, 2014.

[TO15] Takeshi Tsukada and C.-H. Luke Ong. Nondeterminism in game semantics via sheaves. In LICS, 2015.

[Tze11] Nikos Tzevelekos. Fresh-register automata. In POPL, 2011.

[vBMOW05] Franck van Breugel, Michael W. Mislove, Joël Ouaknine, and James Worrell. Domain theory, testing and simulation for labelled markov processes. Theor. Comput. Sci., 2005.

[VKS19] Matthijs Vákár, Ohad Kammar, and Sam Staton. A domain theory for statistical probabilistic programming. PACMPL, (POPL), 2019.

[Wal04] Matthew Wall. Games for syntactic control of interference. PhD thesis, 2004.

[Win86] Glynn Winskel. Event structures. In Advances in Petri Nets, 1986.

[Win13] Glynn Winskel. Distributed probabilistic and quantum strategies. Electr. Notes Theor. Comput. Sci., 2013.

[WPMA19] Aaron Weiss, Daniel Patterson, Nicholas Matsakis, and Amal Ahmed. Oxide: The essence of Rust. 2019.

[YKM14] Kazuhide Yasukata, Naoki Kobayashi, and Kazutaka Matsuda. Pairwise reachability analysis for higher order concurrent programs by higher-order model checking. In CONCUR, 2014.